Lab 2: General-Purpose Timers and Interrupts

Introduction

In this second lab, you will be modifying your work from lab 1 to implement timers and interrupts.

By the end of this lab, you will:

- Have a basic understanding of and experience using general-purpose timers and interrupts.
- Be familiar with the TM4C123 Vector Table and know how to modify it.
- Understand how interrupts can be used, and how to write Interrupt Service Routines.
- Have become even more competent at extracting data from the TM4C's datasheet.

Task 1: General-Purpose Timers

Timers

The TM4C123GH6PM General-Purpose Timer Module (GPTM) provides programmable timers that can be used to count or to drive events such as I/O, communication, or analog to digital conversions. The GPTM is divided into six 16/32-bit GPTM blocks and six 32/64-bit GPTM blocks. Each GPTM block provides two timers (referred to as Timer A and Timer B) that can be configured to operate independently, or they can be configured to be concatenated. When concatenated, they operate as one 32-bit timer (for the 16/32-bit blocks) or one 64-bit timer (for the 32/64-bit blocks).

There are also other timers available on the Tiva C Series microcontrollers such as the System Timer (SysTick) and the PWM timer which can be utilized when required by various application.

For this lab, we will focus on configuring and using General Purpose Timer A. There are various modes that this timer can be configured in; refer to section 11.3.2 of the datasheet for more information. We will first use the timer to blink an LED periodically every second, and therefore, we will start by configuring the timer to use the periodic timer mode.

Initialization and Configuration

Section 11.4 of the datasheet details how to initialize the GPTMs. For this section of the lab, we need to perform the following steps:

For all of these steps, we will use the 16/32-bit Timer 0.

1. Enable the appropriate **TIMERn** bit in the **RCGCTIMER** register.
2. Disable the timer using the **GPTMCTL** register.
3. Select 32-bit mode using the **GPTMCFG** register.
4. Configure the **TAMR** bit of the **GPTMTAMR** register to be in periodic timer mode.
5. Configure the **TACDIR** bit of the **GPTMTAMR** register to count down.
6. Load a value of 16,000,000 into the **GPTMTAILR** to achieve a 1 Hz blink rate using the 16 MHz oscillator.
7. If using interrupts, configure the **GPTMIMR** register. This is not necessary for task 1.
8. Enable the timer using the **GPTMCTL** register.

GPTM Polling

To make use of the timer we just configured, poll the **TATORIS** bit of the **GPTMRIS** register. This will let you know when the timer has counted down and reached 0. Once this happens, clear the bit by writing to the **GPTMICR** register. Then, you're able to return to polling to see when the timer has reached 0 again.

Task 1 Assignments

1. **Change the color of the user LED in a periodic pattern.**
   Repeat the assigned task from lab 1 where you were asked to rotate through all of the colors of the onboard user LED. Change colors at a rate of 1 Hz by polling a GPTM.
2. **Timed Traffic Light**
   Repeat the traffic light assignment from lab 1. Use timers to make the following modifications:
   - Each time a button is pressed, the system responds only if the user holds the button for at least two seconds.
   - When the Start/Stop button is pressed, the system will start in the *stop* state. After a delay of 5 seconds, the system will move to the *go* state. After another 5 seconds, the system will return to the *stop* state. Until a button is pressed, this should repeat indefinitely.
   - When the Pedestrian button is pressed while in the *go* state, the system should immediately transition to the *warn* state, and remain there for 5 seconds before moving to the *stop* state.

Important note: For this lab, you should restructure your program into functions and sub-functions. Functions should as short as possible and complete smaller tasks, keeping code in your main function to a minimum. A suggestion is to create a function to initialize the GPIO, and another to initialize the timer, and call them both from the main function. Another function that might be helpful is one to turn off and on LEDs. This may seem bothersome, but it is a smart coding practice. It will be helpful when your lab requirements get more complicated.

Also note for this lab, **YOU ARE REQUIRED TO USE YOUR OWN HEADER FILES.** Do not include tm4c123gh6pm.h.

Task 2: Interrupts

The Vector Table

The vector table is a table of pointers to routines. These routines are written to handle interrupts and could be used for operations such as Pulse Width Modulation (PWM) or Analog to Digital Conversion (ADC), which you may learn about later. We will be looking at the vector table for the ARM Cortex chip found on the TM4C. There is a standard default vector table that is included each time you compile and upload your code onto the board, but you can also create and modify your own.

The vector table of the TM4C123 LaunchPad is found in Figure 2-6 of the datasheet (page 107). The table here contains the numbers and addresses of each interrupt. The vector table is located at address 0 by default (it can be moved, but that's not our goal here). When you use the debugger in IAR, you can see the vector table contents in the disassembly, from address 0x0 to 0x3c, shown in Figure 1.

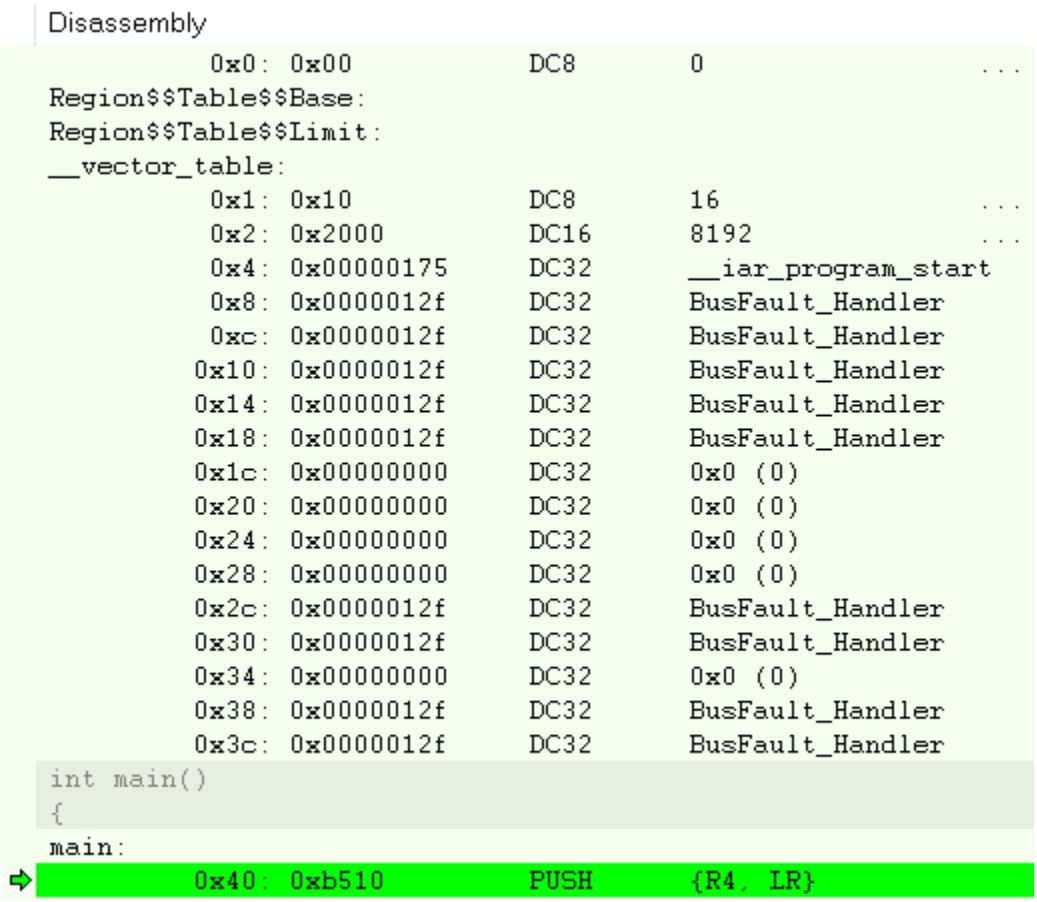

Figure 1: The vector table of the TM4C123 shown in the IAR debugger disassembly

The vector table is implemented in a file named **cstartup_M.c**. The default vector table is located in the IAR installation directory. You will probably find it here:

C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\src\lib\thumb

In this file, you'll find the vector table defined as an array:

```
const intvec_elem __vector_table[] =
{
  { .__ptr = __sfe( "CSTACK" ) },
  __iar_program_start,

  NMI_Handler,
  HardFault_Handler,
  MemManage_Handler,
  BusFault_Handler,
  UsageFault_Handler,
  0,
  0,
  0,
  0,
  SVC_Handler,
  DebugMon_Handler,
  0,
  PendSV_Handler,
  SysTick_Handler

};
```

The order of the array elements matches the Figure 2-6 in the datasheet, with the "Reserved" addresses represented as 0s.

The file also contains the implementation of each of the vector table handlers. We will refer to these as **Interrupt Service Routines** (ISR). The function prototypes for the ISRs is shown as follows:

```
extern void __iar_program_start( void );

extern void NMI_Handler( void );
extern void HardFault_Handler( void );
extern void MemManage_Handler( void );
extern void BusFault_Handler( void );
extern void UsageFault_Handler( void );
extern void SVC_Handler( void );
extern void DebugMon_Handler( void );
extern void PendSV_Handler( void );
extern void SysTick_Handler( void );
```

Our objective is to modify the cstartup_M.c file to add **Interrupt Requests** (IRQ) to the vector table and implement their corresponding ISRs.

Important: do NOT modify the original cstartup_M.c file. Instead, make a copy of it in the directory you're using for lab 2. Then, add the file to your project in IAR. Note that the file is read-only by default, so you need to remove the read-only status of your copy.

Modifying the vector table

We wish to create an ISR to handle the timer we created previously. Table 2-9 in the datasheet (page 104) lists the interrupts of the TM4C123. Timer 0A is interrupt number 19, corresponding to vector number 35. Therefore, we need to append to the vector table to add our ISR, which we will name Timer_Handler:

```
SysTick_Handler,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  0,
  Timer_Handler
```

We also need to add a prototype of the ISR to the vector table to the end of the other prototypes:

```
extern void Timer_Handler( void );
```

And, finally, we need to implement the ISR:

```
#pragma call_graph_root = "interrupt"
__weak void Timer_Handler (void ) { while (1) {} }
```

Now, with your modified vector table added to your project, download and debug your program to the board. You'll notice the vector table has been extended, and our timer handler is located at the bottom, as seen in Figure 2.

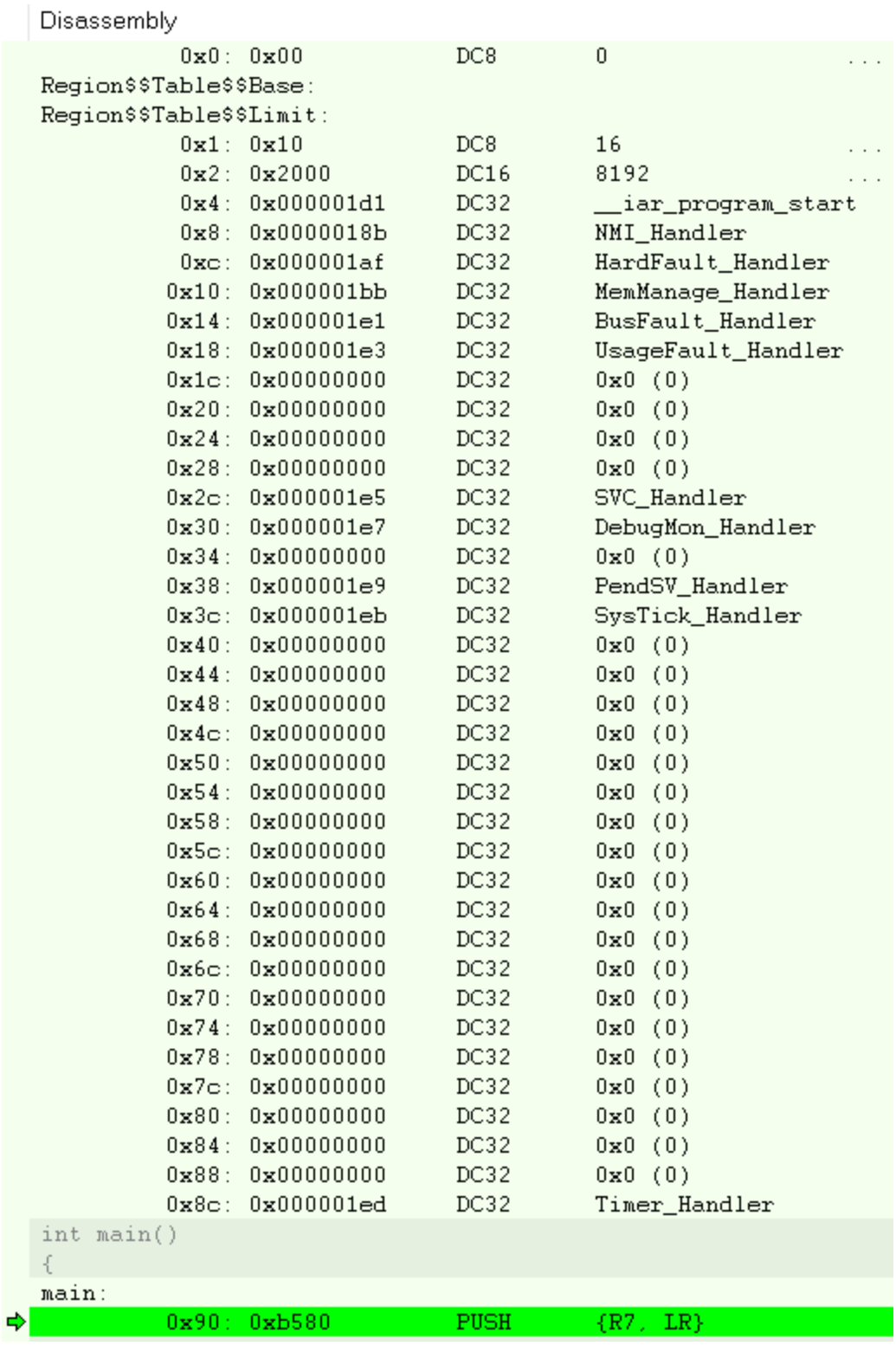

Figure 2: Modified vector table, shown in IAR debugger disassembly

Timer Configuration

To configure Timer 0A to work with your new ISR, you will need to configure the **GPTMIMR** and **EN0** registers. Remember that Timer 0A corresponds to interrupt number 19.

Task 2 Assignments

1. **Change the color of the user LED in a periodic pattern.**
   Repeat the first assigned task from Task 1 using interrupts. Hint: after initializing the timer and LEDs, the ISR can handle everything – your code doesn't need to do anything else.
2. **Controlling the timer from a user switch**
   Using timers, create a program to blink the blue user LED at a rate of 1 Hz. Using interrupts, let the PF0 and PF4 buttons interrupt the program. When PF0 is pressed, the timer should stop working, and instead the red LED should be turned on. When PF4 is pressed, the timer should turn on and blink the blue LED again.
3. **Timed Traffic Light**
   Repeat the second assigned task from Task 1 using interrupts.

Lab Demonstration

Demonstrate your design for the assigned tasks to the TA.

Submission

Submit a short lab report that includes the following sections:

1. Procedure
   - Describe how you approached the assigned tasks and include any relevant diagrams, schematics, etc.
2. Results
   - Describe the results obtained for the assigned tasks. Give a brief overview of the finished product, as it relates to what was asked of you.
   - Include any relevant images, screenshots, etc.
3. Problems Encountered and Feedback
   - Describe any issues you had while completing the lab, and how/if you were able to overcome them.
   - Include any feedback you have about the lab, or any tips and tricks that you learned that may be useful to you in future labs.
   - Include how many hours you spent in completing this lab.
4. Appendix
   - In your Appendix, include any code you wrote in the lab. This should be **formatted** and **commented.**
     A good resource for formatting code for Word documents is found here:
     http://www.planetb.ca/syntax-highlight-word

Additionally, submit your commented .c and .h files.

Grading

Your grade for this assignment will depend on the following:

- Lab demonstration: **50%**
    - Your lab demonstration will be graded based on your completion of the assigned tasks, your ability to answer questions, and your understanding of the course material.
- Lab report: **20%**
    - Your lab report should be short and concise.
- Code comments: **30%**
    - Comment your code! Your code should have function comments and in-line comments where appropriate.